**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по РК №2

Вариант запросов: B

Вариант предметной области: 19

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Нагапетян Валерий |  | Гапанюк Ю. Е. |
|  |  |  |

**Москва, 2023 г.**

**Вариант запросов B.**

# «Деталь» и «Производитель» связаны соотношением один-ко-многим. Выведите список всех деталей, которые начинаются с «Тормоз» и названия их производителей.

# «Деталь» и «Производитель» связаны соотношением один-ко-многим. Выведите список производителей с минимальной стоимостью деталей, отсортированный их по минимальной стоимости.

# «Деталь» и «Производитель» связаны соотношением многие-ко-многим. Выведите список всех связанных деталей и производителей, отсортированный по производителям, сортировка деталей по цене.

**Условия рубежного контроля №2 по курсу ПиКЯП:**

Рубежный контроль представляет собой разработку тестов на языке Python.

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

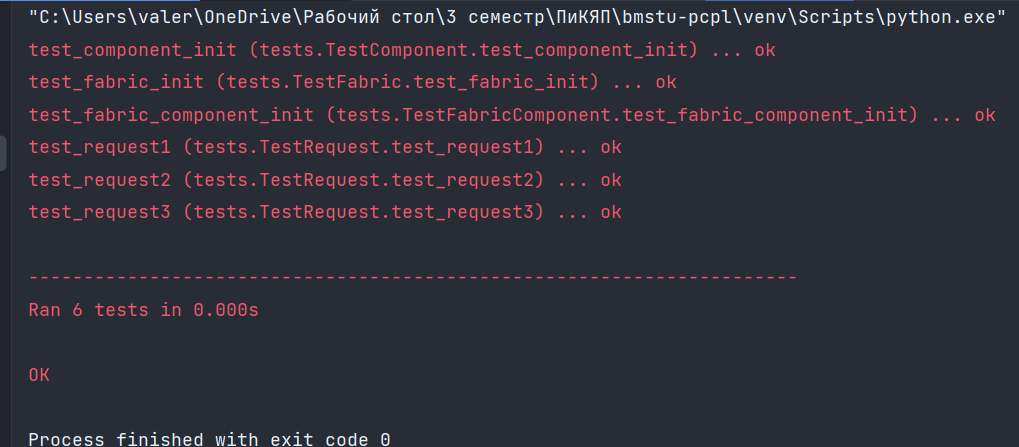
**Листинг программы main.py:**

*from* typing *import* Union, NoReturn  
  
  
*class* Component:  
 *"""Класс Деталь"""  
  
 def* \_\_init\_\_(*self*, id: int, name: str, price: Union[int, float], fabric\_id: int):  
 *self*.\_\_id = id  
 *self*.\_\_name = name  
 *self*.\_\_price = price  
 *self*.\_\_fabric\_id = fabric\_id  
  
 @property  
 *def* id(*self*) -> int:  
 *return self*.\_\_id  
  
 @property  
 *def* name(*self*) -> str:  
 *return self*.\_\_name  
  
 @property  
 *def* price(*self*) -> Union[int, float]:  
 *return self*.\_\_price  
  
 @property  
 *def* fabric\_id(*self*) -> int:  
 *return self*.\_\_fabric\_id  
  
  
*class* Fabric:  
 *"""Класс Производитель"""  
  
 def* \_\_init\_\_(*self*, id: int, name: str):  
 *self*.\_\_id = id  
 *self*.\_\_name = name  
  
 @property  
 *def* id(*self*) -> int:  
 *return self*.\_\_id  
  
 @property  
 *def* name(*self*) -> str:  
 *return self*.\_\_name  
  
  
*class* FabricComponent:  
 *"""Класс детали производителя"""  
  
 def* \_\_init\_\_(*self*, fabric\_id: int, component\_id: int):  
 *self*.\_\_fabric\_id = fabric\_id  
 *self*.\_\_component\_id = component\_id  
  
 @property  
 *def* fabric\_id(*self*) -> int:  
 *return self*.\_\_fabric\_id  
  
 @property  
 *def* component\_id(*self*) -> int:  
 *return self*.\_\_component\_id  
  
  
*def* request1(components: list[Component], fabrics: list[Fabric]) -> list[tuple[str, str]]:  
 response = [  
 (c, f)  
 *for* c *in* components  
 *for* f *in* fabrics  
 *if* c.fabric\_id == f.id *and* c.name.startswith("Тормоз")  
 ]  
 *return* [(c.name, f.name) *for* c, f *in* response]  
  
  
*def* request2(components: list[Component], fabrics: list[Fabric]) -> list[tuple[str, int]]:  
 response = {}  
 *for* fabric *in* fabrics:  
 minimal\_price = min([c.price *for* c *in* components *if* c.fabric\_id == fabric.id])  
 response[fabric.name] = minimal\_price  
  
 response\_items = list(response.items())  
 response\_items.sort(key=*lambda* item: item[1])  
  
 *return* response\_items  
  
  
*def* request3(components: list[Component], fabrics: list[Fabric], fabric\_components: list[FabricComponent]) -> list[tuple[str, str]]:  
 response = [  
 (f, c)  
 *for* fc *in* fabric\_components  
 *for* f *in* fabrics  
 *for* c *in* components  
 *if* fc.fabric\_id == f.id *and* fc.component\_id == c.id  
 ]  
  
 response.sort(key=*lambda* item: (item[0].name, item[1].price))  
 *return* [(fabric.name, component.name) *for* fabric, component *in* response]  
  
  
*def* components\_data\_test() -> list[Component]:  
 *return* [  
 Component(1, "Тормозные колодки", 12000, 1),  
 Component(2, "Фары", 5000, 1),  
 Component(3, "Заднее крыло", 10000, 2),  
 Component(4, "Генератор", 15000, 3),  
 Component(5, "Аккумулятор", 8000, 3),  
 Component(6, "Тормозные диски", 9000, 4),  
 Component(7, "Дворники", 3000, 4)  
 ]  
  
  
*def* fabrics\_data\_test() -> list[Fabric]:  
 *return* [  
 Fabric(1, "АВТОВАЗ"),  
 Fabric(2, "УАЗ"),  
 Fabric(3, "КАМАЗ"),  
 Fabric(4, "УВЗ"),  
 ]  
  
  
*def* fabrics\_components\_data\_test() -> list[FabricComponent]:  
 *return* [  
 FabricComponent(1, 1),  
 FabricComponent(1, 2),  
 FabricComponent(2, 3),  
 FabricComponent(3, 4),  
 FabricComponent(3, 5),  
 FabricComponent(4, 6),  
 FabricComponent(4, 7)  
 ]  
  
  
*def* main() -> NoReturn:  
 components = components\_data\_test()  
 fabrics = fabrics\_data\_test()  
 fabrics\_components = fabrics\_components\_data\_test()  
  
 response1 = request1(components, fabrics)  
 response2 = request2(components, fabrics)  
 response3 = request3(components, fabrics, fabrics\_components)  
  
 print('Запрос №1. Список деталей, которые начинаются с "Тормоз", и их производителей.')  
 *for* (component\_name, fabric\_name) *in* response1:  
 print(f" Деталь: {component\_name} | Производитель: {fabric\_name}")  
 print()  
  
 print("Запрос №2. Список производителей с минимальной стоимостью деталей.")  
 *for* (fabric\_name, minimal\_price) *in* response2:  
 print(f" Производитель: {fabric\_name} | Минимальная стоимость детали: {minimal\_price}")  
 print()  
  
 print(  
 "Запрос №3. Список всех связанных деталей и производителей, отсортированный по производителям. Сортировка деталей по цене."  
 )  
 *for* (fabric\_name, component\_name) *in* response3:  
 print(f" Производитель: {fabric\_name} | Деталь: {component\_name}")  
 print()  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Листинг программы tests.py:**

*import* unittest  
*from* RK2.main *import* \*  
  
  
*class* TestComponent(unittest.TestCase):  
 *def* test\_component\_init(*self*):  
 component = Component(1, "shock absorber", 5871, 1)  
 *self*.assertEqual(component.id, 1)  
 *self*.assertEqual(component.name, "shock absorber")  
 *self*.assertEqual(component.price, 5871)  
 *self*.assertEqual(component.fabric\_id, 1)  
  
  
*class* TestFabric(unittest.TestCase):  
 *def* test\_fabric\_init(*self*):  
 fabric = Fabric(1, "АВТОВАЗ")  
 *self*.assertEqual(fabric.id, 1)  
 *self*.assertEqual(fabric.name, "АВТОВАЗ")  
  
  
*class* TestFabricComponent(unittest.TestCase):  
 *def* test\_fabric\_component\_init(*self*):  
 fc = FabricComponent(1, 2)  
 *self*.assertEqual(fc.fabric\_id, 1)  
 *self*.assertEqual(fc.component\_id, 2)  
  
  
*class* TestRequest(unittest.TestCase):  
 *def* setUp(*self*) -> *None*:  
 *self*.\_components = components\_data\_test()  
 *self*.\_fabrics = fabrics\_data\_test()  
 *self*.\_fabrics\_components = fabrics\_components\_data\_test()  
  
 *def* test\_request1(*self*):  
 expected\_result = [  
 ("Тормозные колодки", "АВТОВАЗ"),  
 ("Тормозные диски", "УВЗ")  
 ]  
 actual\_result = request1(*self*.\_components, *self*.\_fabrics)  
 *self*.assertEqual(actual\_result, expected\_result)  
  
 *def* test\_request2(*self*):  
 expected\_result = [  
 ("УВЗ", 3000),  
 ("АВТОВАЗ", 5000),  
 ("КАМАЗ", 8000),  
 ("УАЗ", 10000)  
 ]  
 actual\_result = request2(*self*.\_components, *self*.\_fabrics)  
 *self*.assertEqual(actual\_result, expected\_result)  
  
 *def* test\_request3(*self*):  
 expected\_result = [  
 ("АВТОВАЗ", "Фары"),  
 ("АВТОВАЗ", "Тормозные колодки"),  
 ("КАМАЗ", "Аккумулятор"),  
 ("КАМАЗ", "Генератор"),  
 ("УАЗ", "Заднее крыло"),  
 ("УВЗ", "Дворники"),  
 ("УВЗ", "Тормозные диски"),  
 ]  
 actual\_result = request3(*self*.\_components, *self*.\_fabrics, *self*.\_fabrics\_components)  
 *self*.assertEqual(actual\_result, expected\_result)  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

**Результат выполнения:**

****